LAB: Software security

Lecturer: Nguyễn Thị Thanh Vân – FIT - HCMUTE

Objective

- Software Security issues
- Sources of Software Vulnerabilities
- Process memory layout
- Software Vulnerabilities - Buffer overflows
  - Stack overflow
  - Heap overflow
- Attacks: code injection & code reuse
- Variations of Buffer Overflow
- Defense Against Buffer Overflow Attacks
  - Stack Canary
  - Address Space Layout Randomization (ASLR)
- Security in Software Development Life Cycle

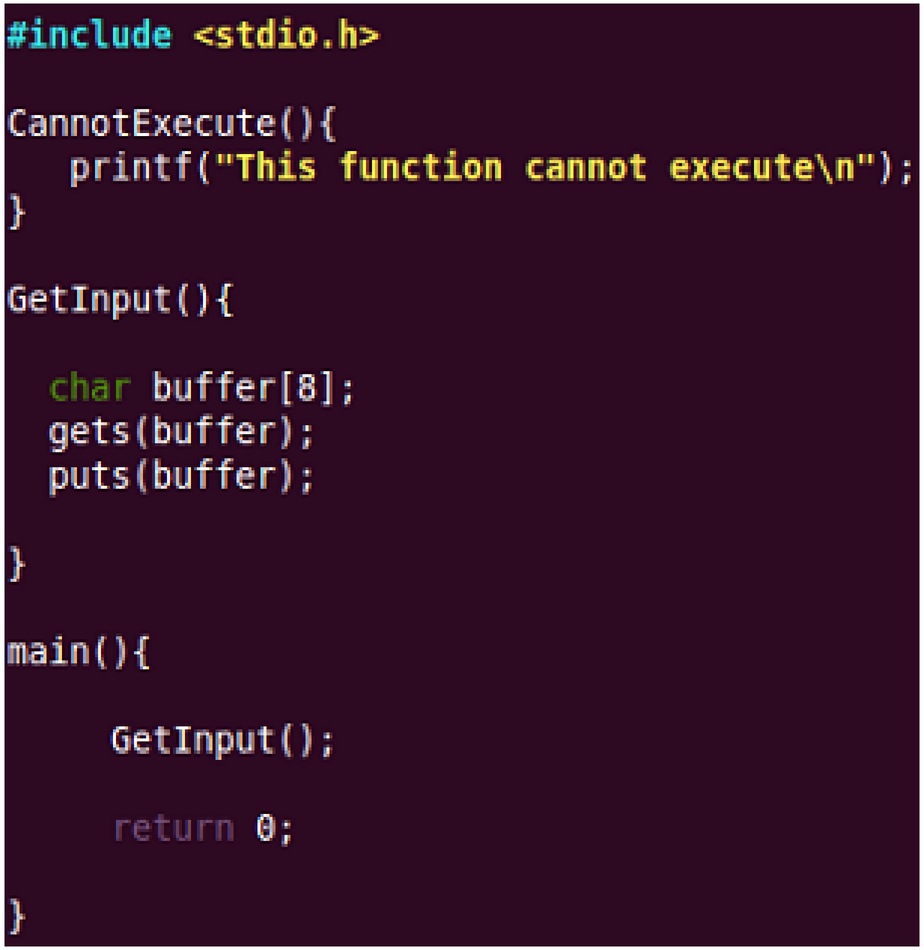
Prepare

- Install a distro of Linux:
  - Ubuntu
  - CentOS
- Install c compiler: gcc(or cc)
  - Check: gcc –v
  - Install: yum install gcc; or apt-get install gcc
- Install gdb:
  - Check: gdb –v
  - Install: yum install gdb; or apt-get install gdb
  - Or: download package gdb and install
    - Download gz, bz2
    - Extract
    - Hit to extracted dir: ./configure; make; make install

Practice

- Follow slides on class
  - Ex1
  - Ex2
- Practice GDB (in file…3.1)
- Practice Buffer overflow (in file …3.2)

Example: Stack Smashing Attack



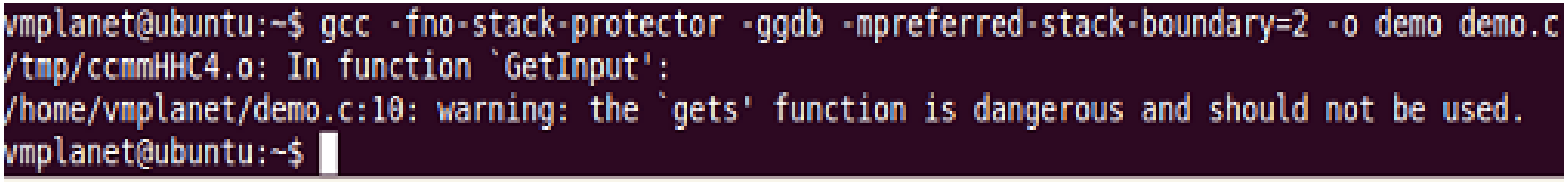
```
#include <stdio.h>

CannotExecute(){
   printf("This function cannot execute\n");
}

GetInput(){

  char buffer[8];
  gets(buffer);
  puts(buffer);

}

main(){

     GetInput();

     return 0;

}
```

Name of the program is demo.c

Assume Little Endian System

Steps

1 Compile with the following options



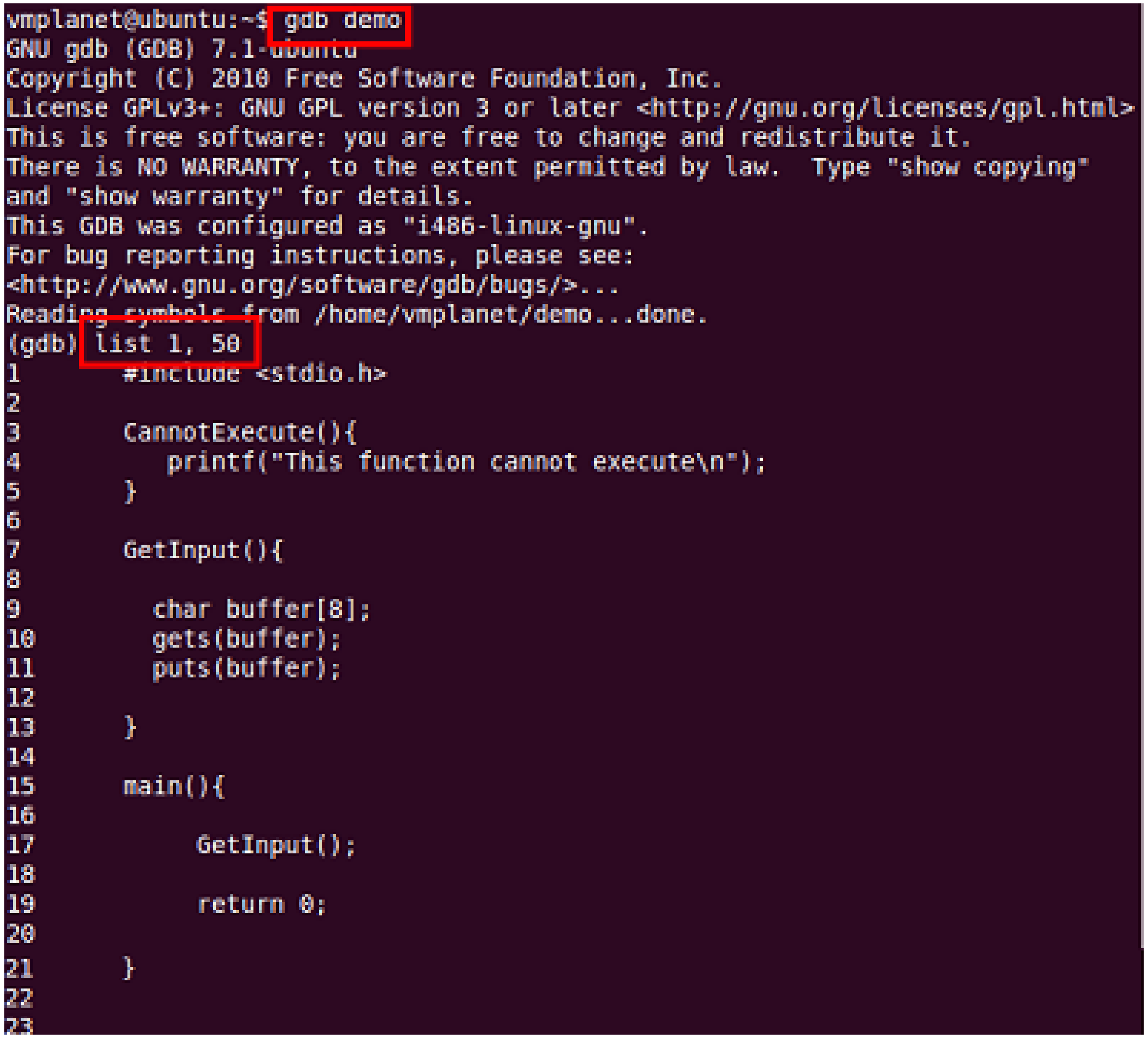
```
vmplanet@ubuntu:~$ gcc -fno-stack-protector -ggdb -mpreferred-stack-boundary=2 -o demo demo.c
/tmp/ccmmHHC4.o: In function `GetInput':
/home/vmplanet/demo.c:10: warning: the `gets' function is dangerous and should not be used.
vmplanet@ubuntu:~$ 
```

2 Start gdb and use the list command to find the line numbers of the different key statements/function calls so that the execution can be more closely observed at these points.

Use list 1,50 (where 50 is some arbitrarily chosen large number that is at least guaranteed to be the number of lines in the program).

In our sample program, we have only 23 lines. So, I could have used list 1, 23 itself.

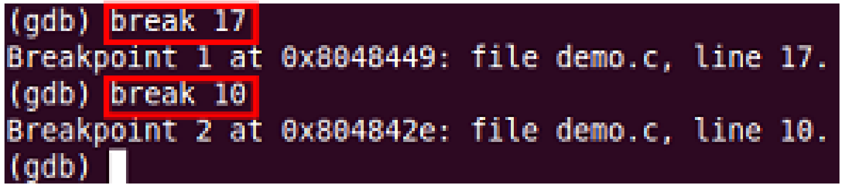
Steps



```
vmplanet@ubuntu:~$ gdb demo
GNU gdb (GDB) 7.1-ubuntu
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/vmplanet/demo...done.
(gdb) list 1, 50
1       #include <stdio.h>
2
3       CannotExecute(){
4          printf("This function cannot execute\n");
5       }
6
7       GetInput(){
8
9         char buffer[8];
10        gets(buffer);
11        puts(buffer);
12
13      }
14
15      main(){
16
17           GetInput();
18
19           return 0;
20
21      }
22
23
```

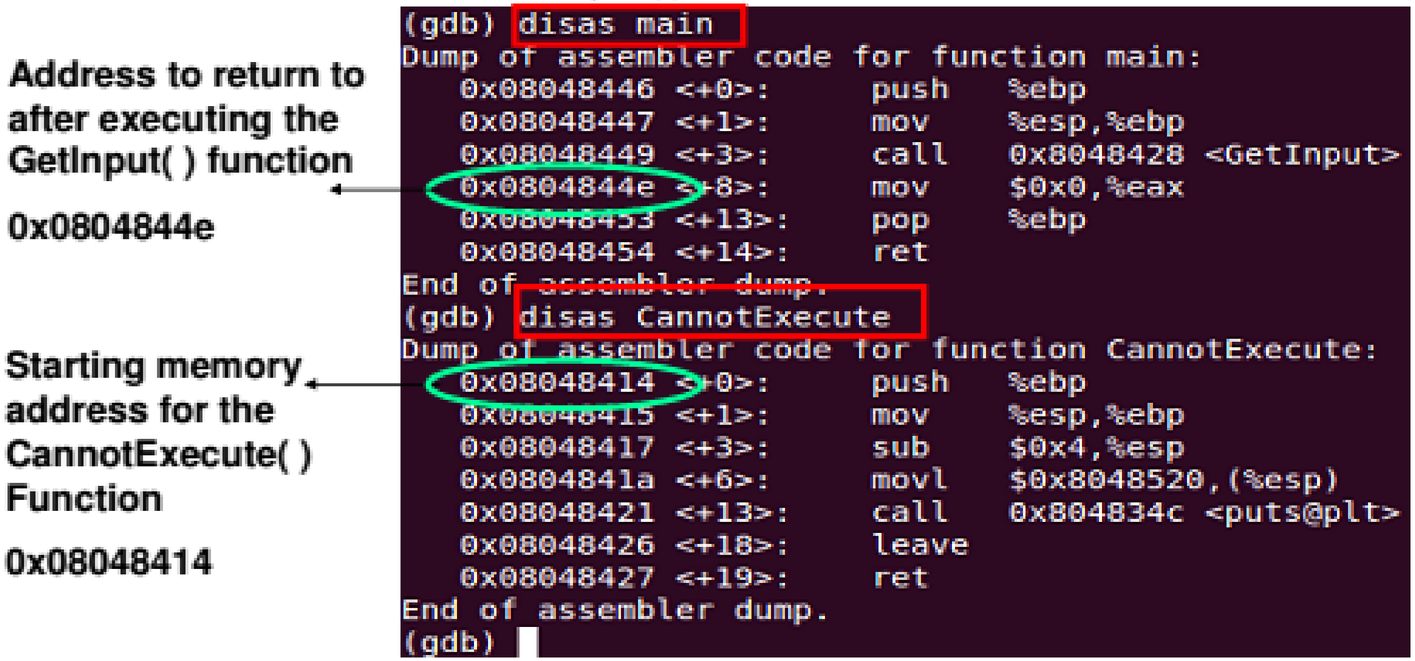
Steps

3 Issue breakpoints at lines 17 and 10 to temporarily stop execution



```
(gdb) break 17
Breakpoint 1 at 0x8048449: file demo.c, line 17.
(gdb) break 10
Breakpoint 2 at 0x804842e: file demo.c, line 10.
(gdb)
```

4 Run the ***disas*** command on the CannotExecute and main functions to respectively find the starting memory address and return address after the return from GetInput( ).



Address to return to after executing the GetInput( ) function

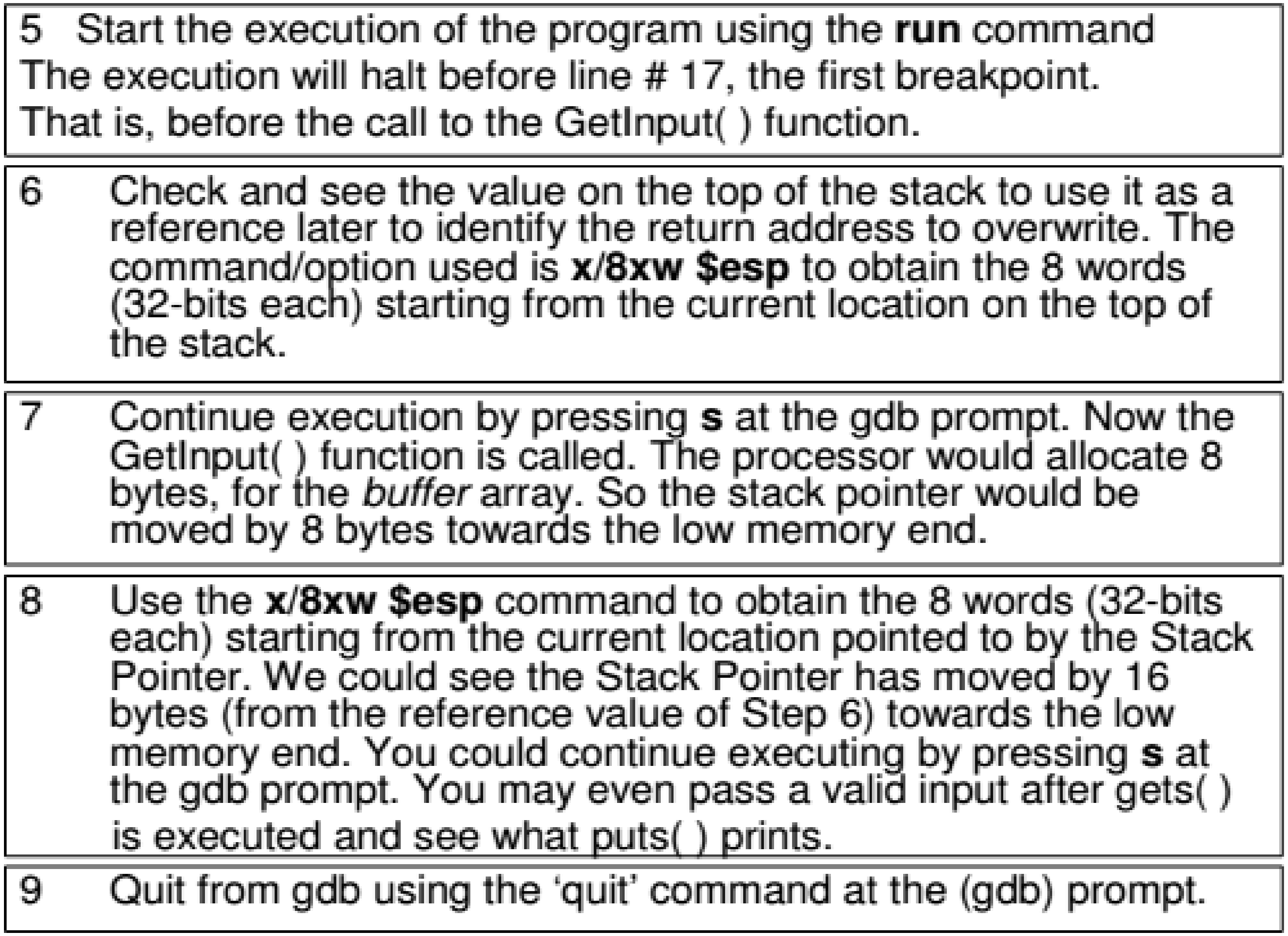
0x0804844e

Starting memory address for the CannotExecute( ) Function

0x08048414

```
(gdb) disas main
Dump of assembler code for function main:
   0x08048446 <+0>:     push   %ebp
   0x08048447 <+1>:     mov    %esp,%ebp
   0x08048449 <+3>:     call   0x8048428 <GetInput>
   0x0804844e <+8>:     mov    $0x0,%eax
   0x08048453 <+13>:    pop    %ebp
   0x08048454 <+14>:    ret
End of assembler dump.
(gdb) disas CannotExecute
Dump of assembler code for function CannotExecute:
   0x08048414 <+0>:     push   %ebp
   0x08048415 <+1>:     mov    %esp,%ebp
   0x08048417 <+3>:     sub    $0x4,%esp
   0x0804841a <+6>:     movl   $0x8048520,(%esp)
   0x08048421 <+13>:    call   0x804834c <puts@plt>
   0x08048426 <+18>:    leave
   0x08048427 <+19>:    ret
End of assembler dump.
(gdb)
```

Steps



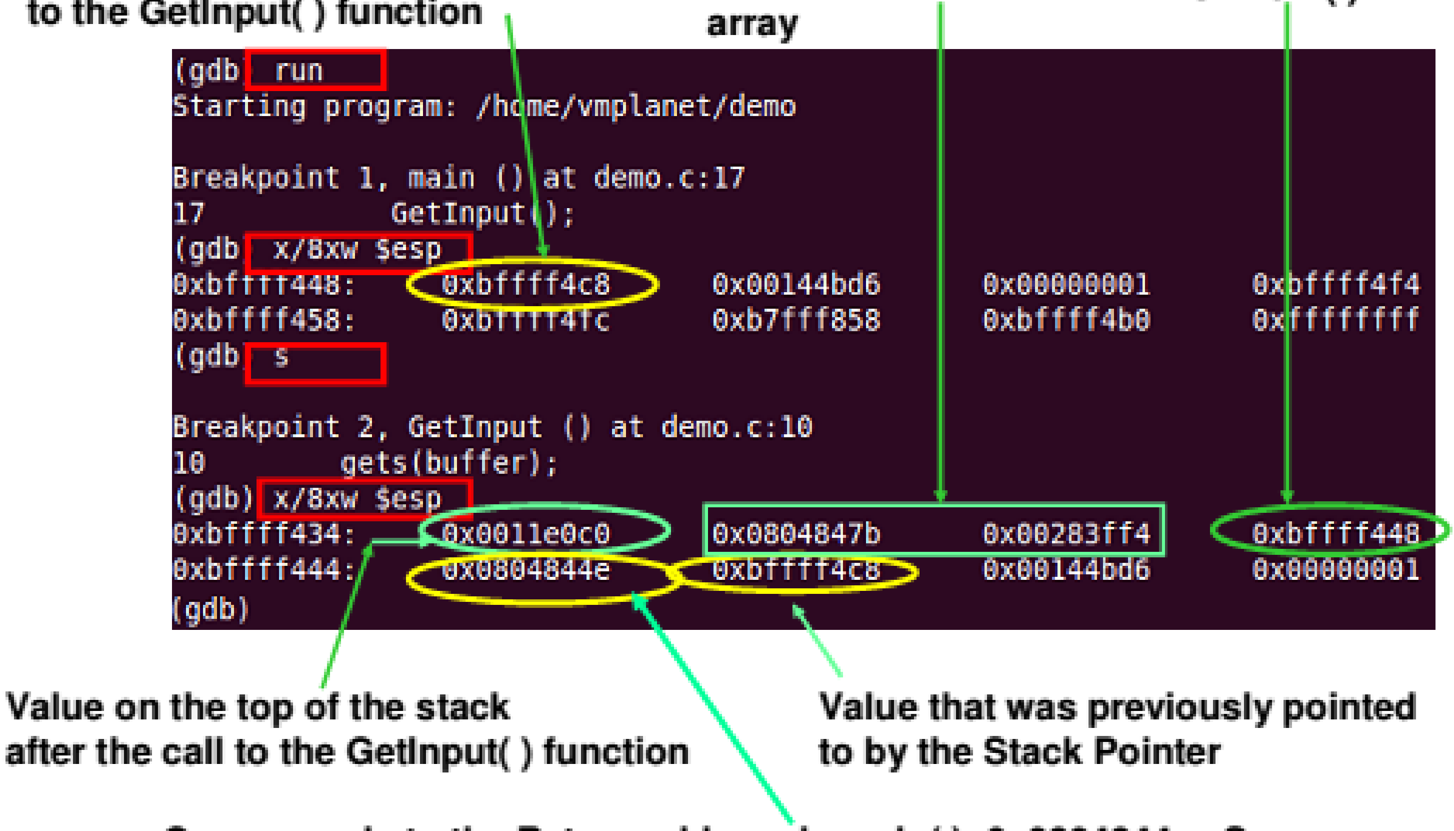
5	Start the execution of the program using the **run** command The execution will halt before line # 17, the first breakpoint. That is, before the call to the GetInput( ) function.
6	Check and see the value on the top of the stack to use it as a reference later to identify the return address to overwrite. The command/option used is **x/8xw $esp** to obtain the 8 words (32-bits each) starting from the current location on the top of the stack.
7	Continue execution by pressing **s** at the gdb prompt. Now the GetInput( ) function is called. The processor would allocate 8 bytes, for the *buffer* array. So the stack pointer would be moved by 8 bytes towards the low memory end.
8	Use the **x/8xw $esp** command to obtain the 8 words (32-bits each) starting from the current location pointed to by the Stack Pointer. We could see the Stack Pointer has moved by 16 bytes (from the reference value of Step 6) towards the low memory end. You could continue executing by pressing **s** at the gdb prompt. You may even pass a valid input after gets( ) is executed and see what puts( ) prints.
9	Quit from gdb using the 'quit' command at the (gdb) prompt.

Steps

Value at the memory address on the top of the stack before the call to the GetInput( ) function

8 bytes of the buffer array

Value of the Frame Pointer for main( )



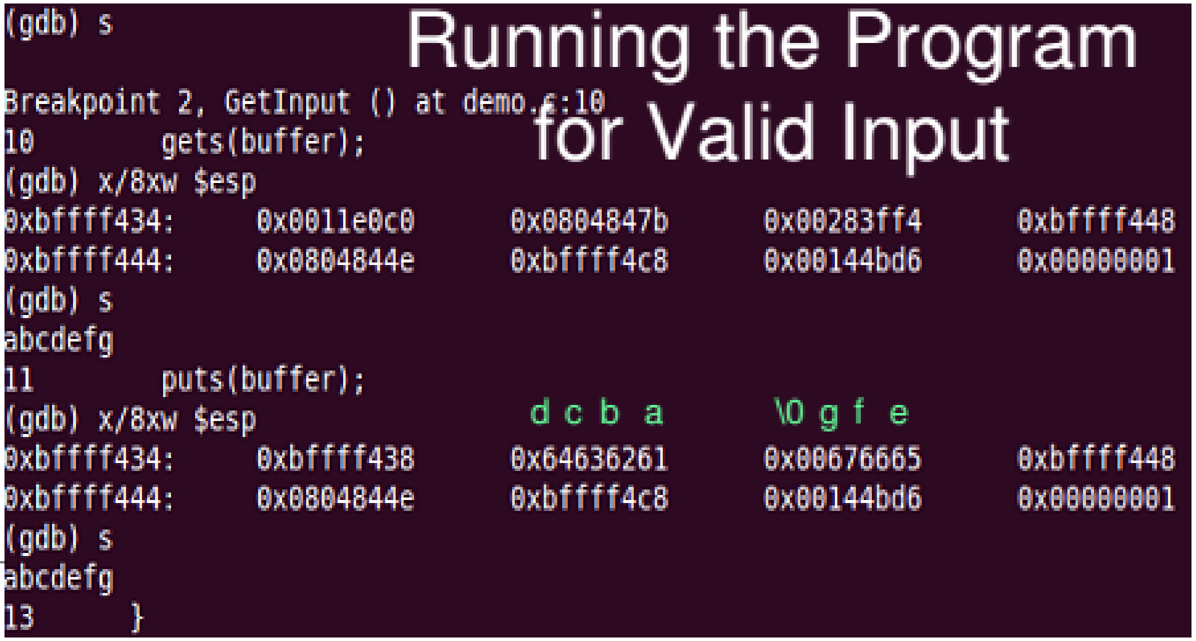
```
(gdb) run
Starting program: /home/vmplanet/demo

Breakpoint 1, main () at demo.c:17
17              GetInput();
(gdb) x/8xw $esp
0xbffff448:     0xbffff4c8      0x00144bd6      0x00000001      0xbffff4f4
0xbffff458:     0xbffff4fc      0xb7fff858      0xbffff4b0      0xffffffff
(gdb) s

Breakpoint 2, GetInput () at demo.c:10
10          gets(buffer);
(gdb) x/8xw $esp
0xbffff434:     0x0011e0c0      0x0804847b      0x00283ff4      0xbffff448
0xbffff444:     0x0804844e      0xbffff4c8      0x00144bd6      0x00000001
(gdb)
```

Value on the top of the stack after the call to the GetInput( ) function

Value that was previously pointed to by the Stack Pointer

Corresponds to the Return address in main( ): 0x0804844e. See the screenshot for Step 4. This is the address that needs to be overwritten with the starting address for the CannotExecute( ) function

Valid input

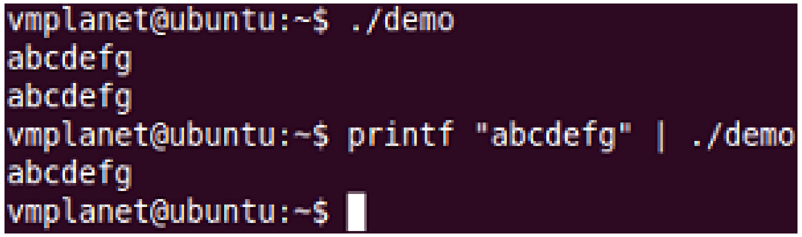


Passing a valid input

Desired output

```
(gdb) s

Breakpoint 2, GetInput () at demo.c:10
10          gets(buffer);
(gdb) x/8xw $esp
0xbffff434:     0x0011e0c0      0x0804847b      0x00283ff4      0xbffff448
0xbffff444:     0x0804844e      0xbffff4c8      0x00144bd6      0x00000001
(gdb) s
abcdefg
11          puts(buffer);
(gdb) x/8xw $esp                d c b a        \0 g f e
0xbffff434:     0xbffff438      0x64636261      0x00676665      0xbffff448
0xbffff444:     0x0804844e      0xbffff4c8      0x00144bd6      0x00000001
(gdb) s
abcdefg
13      }
```

Either way of passing inputs is fine when we pass just printable Regular characters



```
vmplanet@ubuntu:~$ ./demo
abcdefg
abcdefg
vmplanet@ubuntu:~$ printf "abcdefg" | ./demo
abcdefg
vmplanet@ubuntu:~$ 
```

When we want to pass non-printable characters or memory addresses, we need to use the printf option (need to pass them as hexadecimal values)